CS503/CISC850 Development of Scientific Software

Time: **TBA**

Place: **TBA**

Instructor: Diane Kelly,
Department of Mathematics and Computer Science, RMC
G310, 613-541-6000 x6171

Scientific software developers are the traditional pioneers of the field of computing. However, it is well recognized that scientific software developers, both in industry and research, have become isolated, not only from the computing community, but also from each other. There is little communication and little sharing of accumulated knowledge. Secondly the scientists are largely unaware of what the computing community can offer. As a result, there is little transfer of appropriate knowledge between the computing discipline and the scientific disciplines.

It is very important to formalize the exchange of knowledge between the computing community and scientists/engineers developing software. The aim of CS503 is twofold. First, explore for appropriateness of use, software methods and techniques that have been developed either by the computing community or by the scientific community. Second, provide a venue for students in either computing or science/engineering disciplines to broaden their understanding of software techniques and methods, as well as studies of their applicability to scientific software.

Work Load:

3 lectures per week; readings assigned each week for discussion - 10% for participation

Other assignments: (weighting details in a separate handout)

Prepare for and lead the discussion for one topic chosen from the scheduled topics; (details in a separate handout)

Individual research paper and presentation on a topic approved by the instructor (details in a separate handout)

Individual or small team project (details in a separate handout)

At the discretion of the instructor, a student may be required to write a final exam.

Scheduled topics:

Topic	Goal for this topic	Reading list
(1) Software quality as a tool for assessment and improvement	Provide an understanding of the meaning and difficulties associated with quality for scientific software.	Les Hatton, Andy Roberts, "How Accurate is Scientific Software?", IEEE Transactions on Software Engineering, Vol. 20, N0. 10, October, 1994, pp. 785-797 Les Hatton, "The Chimera of Software Quality", IEEE Computer, August 2007, pp.102-104 D.E. Stevenson, "A Critical Look at Quality in Large-Scale Simulations", Computing in Science and Engineering, May-June 1999, pp. 53-63
(2) Risks involved in building scientific software	A look at the social and technical issues that contribute to the risks of building scientific software.	Judith Segal, "Some Problems of Professional End-User Developers", to appear in VL/HCC 2007, 8 pages Douglass E. Post, Lawrence G. Votta, "Computational Science Demands a New Paradigm", Physics Today, January 2005, pp. 35-41 plus letters August 2005 Richard P. Kendall, Douglass E. Post, Jeffrey C. Carver, Dale B. Henderson, David A. Fisher, "A Proposed Taxonomy for Software Development Risks for High-Performance Computing (HPC) Scientific/Engineering Applications", Technical Notes CMU/SEI-2006-TN-039, January 2007, 31 pages
(3) Design and Implementation - I	Architectural styles, design patterns, object oriented analysis and design, and information hiding are examined in terms of the needs of computational scientific software.	Dorian C. Arnold, Jack J. Dongarra, "Developing an Architecture to Support the Implementation and Development of Scientific Computing Applications", in The Architecture of Scientific Software, ed. Ronald F. Boisvert, Ping Tak Peter Tang, Kluwer Academic Publishers, 2000 Ian Foster, Carl Kesselman, "Scaling System-Level Science: Scientific Exploration and IT Implications", IEEE Computer, November 2006, pp. 31-39 Robert B. Laughlin, "The Physical Basis of Computability", Computing in Science and Engineering, May/June 2002, pp. 27-30

(4) Design and Implementation - II	The implementation stage for scientific software is ultimate focus of effort in solving the application domain problem. Here we look at implementation issues including languages, assumptions, constraints, maintainability, and verifiability.	Charles Blilie, "Patterns in Scientific Software: An Introduction", Computing in Science and Engineering, May/June 2002, pp.48-53 (** I don't agree with a lot of what this author says but I can't find a really good paper on design patterns for scientific software!) Viktor K. Decyk, Charles D. Norton, Hanry J. Gardner, "Why Fortran?", Computing in Science and Engineering, July/August 2007, pp. 68-71 Charles Norton, Viktor Decyk, Boleslaw Szymanski, Henry Gardner, "The Transition and Adoption of Modern Programming Concepts for Scientific Computing in Fortran", Scientific Programming, Vol. 15, no. 1, spring 2007, 27 pages
(5) Open Source	Open source is seen as both a significant advance in software development paradigms and as a flavour-of-the-month that is not self-sustaining. We look at what OS means in general and specifically for scientific software development.	Eric S. Raymond, "The Cathedral and the Bazaar", first presented at Linux Kongress, May 1997, www.firstmonday.org/issues/issue3_3/raymond/#d1 Michael W. Godfrey, Qiang Tu, "Evolution in Open Source Software: A Case Study", Proceedings International Conference in Software Maintenance, 2000, pp. 131-142 W. Scacchi, "Understanding the Requirements for Developing Open Source Software Systems", IEE Proceedings – Software, Vol. 149, No. 1, February 2002, pp. 24-39
(6) Development process	Given the goals and needs of scientific software development, we explore the different development processes for their appropriateness and useful practices for scientific software development.	Craig Larman, Victor R. Basili, "Iterative and Incremental Development: A Brief History", IEEE Computer, June 2003, pp.47-56 William A. Wood, William L. Kleb, "Exploring XP for Scientific Research", IEEE Software, May/June 2003, pp.30-36 D.E.Post, R.P.Kendall, "Software Project management and Quality Engineering Practices for Complex, Coupled Multiphysics, Massively Parallel Computational Simulations: Lessons

		Learned from ASCI", The International Journal of High Performance Computing Applications, Vol. 18, No. 4, winter 2004, pp.399-416
(7) Software Inspection	Inspection is recognized as the most cost-effective method of evaluating software. We will look at both traditional definitions of inspection and the newer variations.	Victor Basili, "Evolving and Packaging Reading Technologies", Journal of Systems and Software, Vol. 38, 1997, pp. 3-12 David L. Parnas, David M. Weiss, "Active Design Reviews: Principal and Practices", Proceedings 8th International Conference on Software Engineering, London, August 1985, pp.132-136 Les Hatton, "Testing the Value of Checklists in Code Inspections", www.leshatton.org/Documents/checklists_in_inspectins.pdf, (published web only), January 2007
(8) Testing - overview	Scientists are dealing with software that is difficult to test. This is an overview of testing as described by the computing community. We'll do some comparisons with what scientists may need.	David Talby, Arie Keren, Orit Hazzan, Yael Dubinsky, "Agile Software Testing in a Large-Scale Project", IEEE Software, July/August 2006, pp. 30-37 Marc Eistenstadt, "My Hairiest Bug War Stories", Communications of the ACM, April 1997, Vol. 40, No. 4, pp. 30-37 Glenford J. Meyers, Software Reliability Principles and Practice, Chapter 10: "Testing Principles", A Wiley-Interscience Publication, 1976, pp. 169-195
(9) Testing - techniques	There are a myriad of techniques available. Some of the better-known standard techniques are presented. Discussion of weaknesses and strengths of the different techniques.	W.E. Howden, "Applicability of Software Validation Techniques to Scientific Programs", ACM Transactions on Programming Languages and Systems, Vol. 2, No. 3, July 1980, pp. 307-320 Natalia Juristo, Ana M. Moreno, Sira Vegas, Martin Solari, "In Search of What We Experimentally Know about Unit Testing", IEEE Software, November/December 2006, pp. 72-80 James H. Andrews, Lionel C. Briand, Yvan Labiche, Akbar Siami Namin, "Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria",

		IEEE Transactions on Software Engineering, Vol. 32, No. 8, August 2006, pp. 608-624
(10) Formal specifications and verification	Formal methods have been used for safety critical software, which is often real-time control software. The appropriateness and use of formal methods for scientific computational software is explored.	Daniel M. Berry, "Formal Methods: The very idea, some thoughts about why they work when they work." Electronic Notes in Theoretical Computer Science, Vol. 25, 1999, 13 pages, www.elsevier.nl/locate/entcs/volume25.html Michael Jackson, "What can we expect from program verification?", IEEE Computer, October 2006, pp. 65-71 Spencer Smith, "Systematic Development of Requirements Documentation for General Purpose Scientific Computing Software", 14th IEEE International Conference on Requirements Engineering, 2006, pp.209-218